NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

TESTING A LOW-INTERACTION HONEYPOT
AGAINST LIVE CYBER ATTACKERS

by

Erwin E. Frederick

September 2011

Thesis Advisor:	Neil C. Rowe
Second Reader:	Daniel F. Warren

Approved for public release; distribution is unlimited

Report Documentation Page		*Form Approved* *OMB No. 0704-0188*

Public reporting burden for the collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington VA 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to a penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number.

1. REPORT DATE **SEP 2011**	2. REPORT TYPE **N/A**	3. DATES COVERED **-**	
4. TITLE AND SUBTITLE **Testing a Low-Interaction Honeypot against Live Cyber Attackers**		5a. CONTRACT NUMBER	
		5b. GRANT NUMBER	
		5c. PROGRAM ELEMENT NUMBER	
6. AUTHOR(S)		5d. PROJECT NUMBER	
		5e. TASK NUMBER	
		5f. WORK UNIT NUMBER	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **Naval Postgraduate School Monterey, CA 93943-5000**		8. PERFORMING ORGANIZATION REPORT NUMBER	
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)		10. SPONSOR/MONITOR'S ACRONYM(S)	
		11. SPONSOR/MONITOR'S REPORT NUMBER(S)	
12. DISTRIBUTION/AVAILABILITY STATEMENT **Approved for public release, distribution unlimited**			
13. SUPPLEMENTARY NOTES **The original document contains color images.**			

14. ABSTRACT
The development of honeypots as decoys designed to detect, investigate, and counterattack unauthorized use of information systems has produced an arms race between honeypots (computers designed solely to receive cyber attacks) and anti-honeypot technology. To test the current state of this race, we performed experiments in which we ran a small group of honeypots, using the low-interaction honeypot software Honeyd, on a network outside campus firewall protection. For 15 weeks, we ran different configurations of ports and service scripts, and simulated operating systems to check which configurations were most useful as a research honeypot and which were most useful as decoys to protect other network users. We analyzed results in order to improve the results for both purposes in subsequent weeks. We did find promising configurations for both purposes; however, good configurations for one purpose were not necessarily good for the other. We also tested the limits of Honeyd software and identified aspects of it that need to be improved. We also identified the most common attacks, most common ports used by attackers, and degree of success of decoy service scripts.

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT **unclassified**	b. ABSTRACT **unclassified**	c. THIS PAGE **unclassified**	**SAR**	**89**	

Standard Form 298 (Rev. 8-98)
Prescribed by ANSI Std Z39-18

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE			*Form Approved OMB No. 0704-0188*
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503.			
1. AGENCY USE ONLY *(Leave blank)*	**2. REPORT DATE** September 2011	**3. REPORT TYPE AND DATES COVERED** Master's Thesis	
4. TITLE AND SUBTITLE Testing a Low-Interaction Honeypot against Live Cyber Attackers		**5. FUNDING NUMBERS**	
6. AUTHOR(S) Erwin E. Frederick			
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000		**8. PERFORMING ORGANIZATION REPORT NUMBER**	
9. SPONSORING /MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A		**10. SPONSORING/MONITORING AGENCY REPORT NUMBER**	
11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government. IRB Protocol Number: N/A			
12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution is unlimited		**12b. DISTRIBUTION CODE**	
13. ABSTRACT (maximum 200 words)			

The development of honeypots as decoys designed to detect, investigate, and counterattack unauthorized use of information systems has produced an "arms race" between honeypots (computers designed solely to receive cyber attacks) and anti-honeypot technology. To test the current state of this race, we performed experiments in which we ran a small group of honeypots, using the low-interaction honeypot software Honeyd, on a network outside campus firewall protection.

For 15 weeks, we ran different configurations of ports and service scripts, and simulated operating systems to check which configurations were most useful as a research honeypot and which were most useful as decoys to protect other network users. We analyzed results in order to improve the results for both purposes in subsequent weeks. We did find promising configurations for both purposes; however, good configurations for one purpose were not necessarily good for the other. We also tested the limits of Honeyd software and identified aspects of it that need to be improved. We also identified the most common attacks, most common ports used by attackers, and degree of success of decoy service scripts.

14. SUBJECT TERMS honeypots, Honeyd, honeynet, deception			**15. NUMBER OF PAGES** 89
			16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT Unclassified	**18. SECURITY CLASSIFICATION OF THIS PAGE** Unclassified	**19. SECURITY CLASSIFICATION OF ABSTRACT** Unclassified	**20. LIMITATION OF ABSTRACT** UU

THIS PAGE INTENTIONALLY LEFT BLANK

**TESTING A LOW-INTERACTION HONEYPOT
AGAINST LIVE CYBER ATTACKERS**

Erwin E. Frederick
Lieutenant Commander, Chilean Navy
B.S., Naval Polytechnic Academy, 2001

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

**NAVAL POSTGRADUATE SCHOOL
September 2011**

Author: Erwin E. Frederick

Approved by: Neil C. Rowe, PhD
                Thesis Advisor

                Daniel F. Warren
                Second Reader

                Peter J. Denning, PhD
                Chair, Department of Computer Science

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

The development of honeypots as decoys designed to detect, investigate, and counterattack unauthorized use of information systems has produced an "arms race" between honeypots (computers designed solely to receive cyber attacks) and anti-honeypot technology. To test the current state of this race, we performed experiments in which we ran a small group of honeypots, using the low-interaction honeypot software Honeyd, on a network outside campus firewall protection.

For 15 weeks, we ran different configurations of ports and service scripts, and simulated operating systems to check which configurations were most useful as a research honeypot and which were most useful as decoys to protect other network users. We analyzed results in order to improve the results for both purposes in subsequent weeks. We did find promising configurations for both purposes; however, configurations good for one purpose were not necessarily good for the other. We also tested the limits of Honeyd software and identified aspects of it that need to be improved. We also identified the most common attacks, most common ports used by attackers, and degree of success of decoy service scripts.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

LIST OF FIGURES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF TABLES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF ACRONYMS AND ABBREVIATIONS

CPU	Central Processing Unit
FTP	File Transfer Protocol
HD	Hard Disk
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
IDE	Integrated Drive Electronics
IDS	Intrusion Detection System
IPS	Intrusion Prevention System
MAC	Media Access Control
NetBIOS	Network Basic Input/Output System
NIC	Network Interface Card
OS	Operating System
PCAP	Packet Capture
PCI	Peripheral Component Interconnect
RST	Reset (TCP Flag)
SCSI	Small Computer System Interface
SQL	Structured Query language
SMTP	Simple Mail Transfer Protocol
SSH	Secure Shell
SYN	Synchronize (TCP Flag)
SYSLOG	System Logging
TCP	Transmission Control Protocol
VM	Virtual Machine

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGMENTS

I would like to thank the Chilean and U.S. navies for the privilege of studying at the Naval Postgraduate School. Thank you also to Professor Neil Rowe for his motivation, guidance, and support during my thesis study, and to my second reader Professor Daniel Warren who taught me my the first course in computer security. I also appreciate the support of Ms. Nova Jacobs for her friendly, detailed, and accurate advice during the editing of this thesis.

I am especially thankful to my wife, Karla, for her patience and support, and to my two daughters for inspiring and motivating me every day.

THIS PAGE INTENTIONALLY LEFT BLANK

I.   INTRODUCTION

In the last decade, the development of honeypots—decoys set to detect, deflect, or counterattack an unauthorized use of information systems—has been successful enough that attackers have been forced to develop techniques to detect and neutralize honeypots when they are trying to attack networks. Some of these techniques have been successful, leading some security professionals to think that the use of honeypots is now outdated. However, there are also countermeasures against this anti-honeypot technology.

A powerful and flexible tool that is freely available to deploy multiple honeypots is Honeyd (Honey daemon), developed by Security expert Niels Provos [1]. It allows a user to set up and run multiple virtual hosts on a network with services and specific operating systems running. According to its creator, Honeyd could be used for two purposes: as a honeypot, attracting attackers that later could be traced, analyzed, and investigated, and as decoy or distraction, hiding real systems in the middle of virtual systems. The purpose of this study is to analyze how useful Honeyd is for both purposes, and to assess which actions or countermeasures could be useful to improve its performance against possible attackers.

We set up an experiment using a small network on the NPS campus that is not protected by the campus firewall. We ran a group of honeypots created with the aforementioned software and tested them in different runs with different configurations. During the experiment, we analyzed results week by week to identify the best configuration of Honeyd for both research and decoy purposes. We tried to test as many features of Honeyd as possible, such as simulation of open, closed, or filtered ports, and emulation of operating systems at TCP/IP stack level, service scripts associated to certain well-known ports. In order to create a credible set of virtual machines, we also tested small details like changes in the MAC addresses, set drop rates, set uptime, and the use of proxy and tarpit capabilities to create a credible set of virtual machines.

1

In Chapter II, we provide background for this thesis. In Chapter III we describe the applications and software used to set and analyze the results of the experiment. In Chapter IV, we describe the methodology applied to execute and analyze the experiments in this study. In Chapter V, we analyze results obtained in the experiments: alerts, operating systems emulation, ports attacked, service scripts, Honeyd as a honeypot, and Honeyd as decoy. In Chapter VI, we state conclusions obtained in this study and possible future work. Three appendices provide details of the configurations used each week, the text of the code and commands used, and an analysis of the Nmap operating system detection in relation to Honeyd.

II.   PREVIOUS WORK AND BACKGROUND

A.   HONEYPOTS

The concept of warfare in cyberspace is very similar to that of conventional warfare.

Understanding our capabilities and vulnerabilities, and those of our adversaries, allows us to create better defensive and offensive plans. Before 1999, there was very little information about cyber-attacker threats and techniques. Although there were some previous attempts to obtain information about attackers, the creation of the Honeynet Project [2] was the answer to that lack of knowledge. This project is an international nonprofit research organization that collects and analyzes cyber-attacks using a creative-attack data collection tool, the honeypot.

A honeypot is a trap set to detect, analyze, or in some manner counteract attempts of unauthorized use of information systems. Generally, it consists of a computer, data, or network site which seems to contain information or resources of value to attackers, but is actually isolated, protected, and monitored.

The value of a honeypot lies in the fact that its use is unauthorized or illicit [2] because it is not designated as a production component of an information infrastructure. Nobody outside the creator of the honeypot should be using or interacting with honeypots; any interaction with a honeypot is not authorized and is therefore suspicious. Because of this, there are no false positives.

1.   Variations of Honeypots According to Their Interaction Level

There are two main categories of honeypots: Low-interaction and high-interaction [3].

Low-interaction honeypots are passive, and cyber attackers are limited to emulated services instead of actual operating systems. They are generally easier

to deploy and pose minimal risk to the administrators. Examples of low-interaction honeypots are Honeyd, LaBrea Tarpit, BackOfficer Friendly, Specter, and KFSensor.

High-interaction honeypots provide working operating systems and applications for attackers to interact with. They are more complex and serve as better intelligence-collection tools. However, they pose a higher level of risk to the administrator due to their potential of being compromised by cyber attackers, as for instance, with the use of compromised honeypots to propagate other attacks. Examples are the Symantec Decoy Server (formerly ManTrap) and honeynets as an architecture (as opposed to a product or software).

Table 1 summarizes honeypots according to their interaction level.

Low-interaction	High-interaction
Honeypot emulates operating systems, services and network stack.	Full operating systems, applications, and services are provided.
Easy to install and deploy. Usually requires simply installing and configuring software on a computer.	Can be complex to install and deploy (although commercial versions tend to be simpler).
Captures limited amounts of information, mainly transactional data and some limited interaction.	Can capture far more information, including new tools, communications, and attacker keystrokes.
Minimal risk of compromise, as the emulated services control what attackers can and cannot do.	Increased risk of compromise, as attackers are provided with real operating systems with which to interact.

Table 1.   Honeypots according to interaction level

2.   Types of Honeypots According to Their Purpose

Honeypots can be deployed as production or research systems [3]. When deployed as production systems, typically in an enterprise or military network, honeypots can serve to prevent, detect, bait, and respond to attacks. When deployed as research systems, typically in a university or institute, they serve to collect information on threats for analysis, study, and security enhancement.

3.   Types of Honeypots According to Their Implementation

Another distinction exists between physical and virtual honeypots [3]. Physical means that the honeypot is running on a real machine, suggesting that it could be high-interaction and able to be compromised completely. Physical honeypots are expensive to maintain and install, making them impractical to deploy for large address spaces.

Virtual honeypots use one real machine to run one or more virtual machines that act as honeypots. This allows for easier maintenance and lower physical requirements. Usually VMware and User-mode Linux (UML) are used to set up these honeypots.

While reducing hardware requirements for the administrators, virtual honeypots give cyber attackers the perspective of independent systems in networks. This reduces the cost of management of the honeypots for production and research, compared to physical honeypots. There are, however, disadvantages. The use of the virtual machines is limited by the hardware virtualization software and the host operating system. The secure management of the host operating system and virtualization software has to be thoroughly planned and executed in order to prevent cyber attackers from seizing control of the host system, and eventually the entire honeynet. It is also easier to fingerprint a virtual honeynet, as opposed to honeynets deployed with real hardware, by the presence of virtualization software and signatures of the virtual hardware

emulated by the virtualization software. Cyber attackers may potentially identify these signatures and avoid these machines, thereby defeating the purpose of deploying the honeynet.

4.   Types of Honeypots According to Their Side

The last distinction is between server-side and client-side honeypots [3]. Traditional, server-side honeypots are servers which wait passively to be attacked, possibly offering bait. Client honeypots, by contrast, are active devices in search of malicious servers or other dangerous Internet locations that attack clients. The client honeypot appears to be a normal client as it interacts with a suspicious server and then examines whether an attack has occurred. The main target of client honeypots is Web browsers, but any client that interacts with servers can be part of a client honeypot, including SSH, FTP, and SMTP.

Examples of client honeypots are HoneyC, HoneyMonkey, HoneyWare, and HoneyClient.

5.   Honeynets

The value of honeypots can be increased by building them into a network; two or more honeypots on a network form a honeynet [2]. Integrating honeypots into networks can provide cyber attackers a realistic network of systems to interact with, and permits defenders a better analysis of distributed attacks.

6.   Monitoring Tools in a Honeypot

Honeypots typically contain a set of standard tools, including a component to monitor, log, collect, and report the intruder's activity inside the honeypot. The goal is to capture enough data to accurately recreate the events of the honeypot. Data collection can be done in many ways, the most important of which are:
- Honeypot log files
- Packet sniffing (network sniffing or intrusion detection systems)
- Keystroke logging (or keylogging)
- Snapshot software
- Firewall logs

One example of a data capture tool used in honeypots is Sebek. It is an open-source tool whose purpose is to capture from a honeypot as much information as possible of the attacker's activities on the host by intercepting specific system calls, or syscalls, at the kernel level. Sebek takes sophisticated measures to conceal itself, because honeypot monitoring software needs to function as stealthily as possible, so the intruder cannot detect it. Otherwise, the game is over and the honeypot defeated.

As part of the defense-in-depth approach to information security (multiple layers of security controls), and a critical part of honeypot architecture, intrusion detection systems are deployed to detect potential incoming threats based on signature sets or anomalies. Although they are passive, they can overwhelm administrators with alerts instead of responses or actions against detected attacks. To address this problem, intrusion prevention systems can be used with higher thresholds for alerts; they extend the detection capability of IDS to include automated controls in response to cyber-attacks. For instance, they can ignore, block, or modify packets, preventing the success of the exploit. This active capability, however, comes at a cost to the performance of protected networks or systems. Snort is probably the most popular and well-known intrusion-detection system. It is useful in disabling attacks on a honeypot and for later analysis of the data, with the goal of detecting and understanding cyber-attacks against honeypots.

B.   ANTI-HONEYPOT TECHNOLOGY

When security professionals started to include honeypots and honeynets in their arsenal for information defense, cyber attackers reacted by creating tools to detect or disable honeypots. The use of this anti-honeypot technology means that honeypots were affecting the activities of attackers [4].

If an attacker detects a honeypot, most of the time that attacker will avoid it and go to another place. But there is the risk that an attacker could compromise the honeypot and use it to attack other computers on the local network or Internet. The attacker could also try to disable it, delete the data, format the hard

drive, or post its address on hacker websites to prevent other attackers from begin ensnared by it. In any case, it results in the honeypot's defeat.

Most attackers will not bother to compromise a honeypot; however, if the honeypot is a high-priority attack target like a military command-and-control system, and the attacker is a foreign country, manipulation of that honeypot might be desirable. To accomplish such manipulation, several techniques and tools useful to cyber attackers for footprinting or analyzing systems can be reused or adapted. Some of these tools can detect suspicious environments like virtual machines, keyloggers, and debuggers. Additionally, most software used to build and run honeypots has distinguishable characteristics that give attackers clues, such as recognizable directory and file names. User-mode Linux and VMware might be detected in this way.

Another approach to identifying honeypots is to experiment with detecting data collection tools like Sebek. For example, it is possible to detect Sebek by measuring execution time of the read() system call; excessive delays in the execution of some processes or a higher load than normal in a CPU are also good hints. Specific requests and responses to corrupted packets could give a clear warning to the attacker of the presence of Honeyd or—if there are more active responses—LaBrea Tarpit.

There is also commercial honeypot-detection software available, such as Send-Safe Honeypot Hunter. This tool opens a local fake e-mail server on port 25 (SMTP) and asks each proxy to connect back to itself. If the proxy claims that a session is OK when there is no related incoming session, a possible honeypot is detected.

Table 2 lists some honeypots, their associated characteristics, and their potential exploits.

Honeypot/Honeynet	Typical Characteristics	Methods for Detecting the Honeypot
BackOfficer Friendly	Restricted emulation of services and responses	Send different requests and verify the consistency of responses for different services.
LaBrea Tarpit	TCP window size 0; bogus MAC address	Check persistent TCP window size 0 and MAC address (0:0:0:f:ff:ff).
Honeyd	Signature based responses; same clock for every host	Send a mixture of legitimate and illegitimate traffic, with common signatures recognized by targeted honeypots. Analyze timestamps of the hosts.
Snort IPS	Modification actions; suspicious packets could be dropped or modified.	Send different packets and verify the existence and integrity of response packets.
Virtual Honeynet (VMware)	Virtualization and system files	Detect virtual hardware by name and VMware MAC addresses. Probe for existence of VMware.
Active tcpdump session or Sebek	Logging processes	Scan for active logging process or increased round-trip time (for instance, due to read() in Sebek-based honeypots).

Table 2.   Characteristics of some honeypots and ways to detect them

THIS PAGE INTENTIONALLY LEFT BLANK

III. DESCRIPTION OF THE APPLICATIONS

We used several applications to implement the honeypots and to analyze the results: VMware, Honeyd, Snort, Microsoft Log Parser, Wireshark, Security Onion and Fedora 14.

We will describe the applications used in the implementation, with a quick analysis of the methods to detect them, some countermeasures, and finally the software used to analyze the results.

A.   HONEYD

Honeyd is an open-source program released under GNU General Public License that allows a user to set up and run multiple virtual hosts on a network. These virtual hosts can be configured to mimic several different types of servers, allowing the user to simulate many different computer-network configurations. The hosts can be configured to run arbitrary services, and their personality can be adapted so that they appear to be running certain operating systems. Honeyd enables a single host to claim multiple IP addresses. In this way, Honeyd deters adversaries by hiding real systems in the middle of virtual systems.

This daemon software offers several interesting features: It is possible to Ping the virtual machines, or to run a traceroute to find their forwarding packets. Any type of service on the virtual machine can be simulated according to a simple configuration file. Instead of simulating a service, it is also possible to proxy it to another machine, even to the source. The different personalities simulate operating systems at TCP/IP stack level; this configured personality is the operating-system fingerprint that scanning tools like Nmap (Network Mapper) or Xprobe would return.

Although Honeyd is considered a low-interaction honeypot, it has powerful features to run services through scripts that could be configured to go beyond simple port listening and give responses to intruders. In this way, we can increase the level of interaction of the honeypot. Honeyd can be used to create a virtual honeynet or for general network monitoring. It supports the creation of a

virtual network topology, including dedicated routes and routers. The protocols can simulate latency and packet loss to make the topology seem more realistic.

Honeyd software provides two types of logs that are useful to analyze. Network packet-level logging gives an overview or details of what kind of traffic the honeypots receive, and system logging gives more detailed information about the ongoing traffic. Honeyd can be used for two purposes: distracting potential hackers or catching them in a honeypot. Either way, the hackers will be slowed down and subjected to analysis.

Unfortunately, Honeyd has not been updated recently and some features, like operating-systems fingerprinting, do not work well with the later versions of Nmap and Xprobe.

1.   Detection of Honeyd

Honeyd software running on a computer, or the virtual hosts created by Honeyd, could be detected in several ways.

One method is to flood one honeypot with pings or another CPU intensive process. This honeypot machine will use its resources to respond to this request, and as a consequence, all other simulated machines will become slower.

Another possible method is related to time and latency. Apart from the fact that the responses in the simulated systems in the honeynet will always be a little slower than a real system, we could compare clock timestamps of several different components of the net. Normally, every computer will have a slightly different timestamp because their hardware is different. With Honeyd, the timestamps will be more consistent.

Another way to detect Honeyd, is to analyze the responses of the machines to some uncommon packets and try to find discrepancies on the responses. For Honeyd, this happens when a TCP packet, with SYN and RST flags, is probed to an open port. Honeyd will send a reply, while most other machines will not.

Another method to detect, and maybe attack, Honeyd is through packet fragmentation. This method exploits a vulnerability related to the way Honeyd

reassembles fragmented packets. Honeyd checks the source address, destination address, and identification number but not the protocol number. An adversary could send a carefully prepared fragmented packet with mixed protocols that when reassembled by Honeyd could produce a reply packet or execute some attack, whereas normal operating systems would just discard them.

To prevent detection of Honeyd, countermeasures are periodically added to new versions of the software. For example, versions starting from 0.8 solved the clock timestamp problem by providing a different clock skew (timing difference) to each operating system and each virtual honeypot. Additionally, the wrong replies to TCP packets with SYN and RST flags are now patched.

B.   VMWARE

VMware software provides a virtualized set of hardware to the guest operating system. VMware software virtualizes the hardware for a video adapter, a network adapter, and hard disk adapters to give the appearance of an x86 hardware platform. This allows the installation of almost any operating system, while the host provides pass-through drivers for guest USB, serial, and parallel devices. In this way we could run, for example, a guest Linux OS over a Windows OS host.

The virtualization capabilities of VMware software give us an easy way to develop a virtual high-interaction honeypot.

A disadvantage of VMware is that it is relatively easy to detect a VMware machine in several ways.

By default, the MAC address of NIC will be 00:0C:29, 00:50:56, or 00:05:69, the MAC addresses assigned to the vendor VMware by the IEEE. With these restrictions, if the attacker is in the same network, the MAC address will be immediately detected.

The names of IDE and SCSI devices (HD and CDROM) are clearly related to VMware: VMware Virtual IDE Hard Drive, NECVMWar VMware IDE CDER10, and VMware SCSI Controller.

The PCI vendor string and device ID of video adapter, VMware Inc PCI Display Adapter, is visible. Finally, the I/O backdoor in port 0x5658 (22104 in decimal) that can be used to configure VMware during runtime is visible.

1.   Countermeasures against VMware Fingerprinting

There are ways to prevent an attacker from easily detecting the VMware machine or a virtual environment.

There are hex editors that could be used to edit the VMware binary file, "vmware-vmx.exe". We could search for Virtual IDE Hard Drive or Virtual IDE CDROM, and change them to names more appropriate to hide the VMware application. In Linux, this can also be done automatically by using scripts that are made to patch VMware. One such script was made by Kostkya Kortchinsky, a member of the French Honeynet Project. This Linux script gives the option to change the name of the IDE devices (HD and CDROM), SCSI devices (HD and CDROM), PCI vendor and device ID of the video adapter, and the I/O backdoor used by VMware.

An appropriate configuration of the OS could prevent the VM system from being fingerprinted and detected. For example, we need to give each virtual machine enough main memory to be credible, such as 512 MB or above. This change could be done through the VMware Virtual Machine Control Panel. Also, we should change VMware MAC address because the default MAC address assigned by VMware always starts with 00:0C:29, 00:50:56, or 00:05:69.

Operating systems or VMware provide a way to change the MAC address, but we need to be careful to match the numbers to an existing vendor that also is related with changed names of other devices.

C.   SNORT

Snort is a free, cross-platform, open-source network intrusion prevention system and network intrusion detection system, created by Martin Roesch, a respected authority on intrusion detection and prevention technology.

Snort's network-based intrusion detection system has the ability to perform real-time traffic analysis and packet logging on Internet Protocol (IP) networks. Snort performs protocol analysis, content searching, and content matching. The program can also be used to detect probes or attacks, including operating system fingerprinting attempts, common gateway interface, buffer overflows, server message block probes, and stealth port scans.

Snort can be configured in three main modes: sniffer, packet-logger, and network intrusion detection. In sniffer mode, the program will read network packets and display them on the console. In packet-logger mode, the program will log packets to the disk. In intrusion-detection mode, the program will monitor network traffic and analyze it against a defined set of rules. The program could then perform a specific action based on what has been identified.

The rules we used to run Snort were the Sourcefire Vulnerability Research Team (VRT) rules, which are the official rules available for the program. We used the latest VRT rules that were available free to registered users, rules an average of 30 days old when released.

The software provides a detailed alert log, which can be shown in different formats, like a text file or a pcap file, which store the packets associated with the alerts so they can be analyzed with software like Wireshark.

D.   WIRESHARK

Wireshark is a free, open-source packet analyzer. It is used for network troubleshooting, analysis, software and communications protocol development, and education. This software was originally named Ethereal, but it was renamed Wireshark in May 2006.

Wireshark works in a similar way to tcpdump, but with a graphical front-end, plus several sorting and filtering options.

E.   MICROSOFT LOG PARSER

The Microsoft Log Parser is a powerful and very flexible command-line tool that provides universal query access to text-based data such as log files, Extensible Markup Language (XML) files, comma-separated values (CSV) files, and tab-separated values (TSV). It also provides universal query access to key data sources on the Windows operating system such as the Event Log, the Registry, the file system, and the Active Directory. The results of the query can be custom-formatted in text-based output, or they can be exported to targets like SQL, SYSLOG, or Excel.

F.   SECURITY ONION

Security Onion is free distribution created by security expert Doug Burks, which could be either used as a LiveDVD or installed in the hard drive as a virtual machine. It contains software used for installing, configuring, and testing intrusion detection systems based on Xubuntu 10.04 Operating System and contains Snort, Suricata, Sguil, Xplico, Nmap, and many other security tools specially compiled for use in intrusion detection. According to its creator, the software is hardened for its security function.

G.   FEDORA 14

Experiments were conducted in an operating system based on Red Hat Linux Fedora 14 (Laughlin). This was the last version available at the beginning of this study.

IV.   METHODOLOGY

A.   OBJECTIVES

The objective of the main experiment was to deploy a honeynet easily accessible to the Internet, and which could be scanned and attacked. We tried to maximize the interaction with possible attackers, meaning to maximize the number, variety, and duration of attacks. If this is the case, then the honeypots are more successful and difficult to detect or avoid. To do this, machines were simulated using the software Honeyd. The analysis of the number of attacks on them can be compared with the attacks on other hosts of the network, giving us a good idea about how effective the honeypots created with this software were in hiding or protecting the real systems.

We attempted to find the answers to the following questions:

a) How did our simulated network look from the outside?

b) Were the emulated operating systems well simulated by Honeyd?

c) What attacks did the network receive?

d) Did we receive more attacks using Honeyd than without?

e) Did Honeyd attract attacks, diverting them from the real systems?

f) Were there differences in the number or kinds of attacks between the emulated operating systems, protocols, ports, or services?

g) Did the real laptop (Windows XP) and the VM (Fedora 14 or Security Onion) running Honeyd get compromised?

h) What can we do to make the simulated hosts more credible?

i) Could Honeyd be useful in a production environment?

B.   THE EXPERIMENT

The experiment was run at the Naval Postgraduate School campus using a single laptop running Windows XP as host; the laptop also ran first a Fedora 14 Virtual Machine and later a Security Onion VM, both using VMware as guests. This laptop was connected by Ethernet to one port associated to a special network. This network, known as PacBell network (63.205.26.64/27), is a small network at NPS that is outside the protection of the main firewall. It has 32 IP addresses, of which around 10 of them are normally used. We used seven other IP addresses for this experiment, all of them monitored and working as honeypots.

We used a hub to connect to the network because the experiment coexists with other tests and honeypots. One of the latter is a real host running Windows XP that can be considered part of our experiment, although it is production system.

On the Windows machine we installed Snort 2.9 with the VRT rules. Snort was configured to log comma-separated values (CSV) files, and create tcpdump files for the alerts, which could be read with Wireshark. On the Fedora virtual machine we installed Honeyd 1.5c, which generated its own packet and system logs. At the beginning of the experiment and between every run, both machines were updated and patched to harden them against possible attacks. Snort was also updated when a new set of rules available for registered users was released.

We tried different configurations, each of which ran for approximately one week. We used the following criteria to make changes:
-        In general, we went from simpler to more complex.
-        Using previous results, we discarded the less successful configurations in terms of amount of traffic and number and variety of alerts.
-        We changed the relation between IP addresses and the operating systems randomly.

- A couple of times we also ran the experiment without honeypots or even without the guest operating system to study the normal traffic on the network.

Figure 1 shows a diagram of the network architecture used in the experiment.
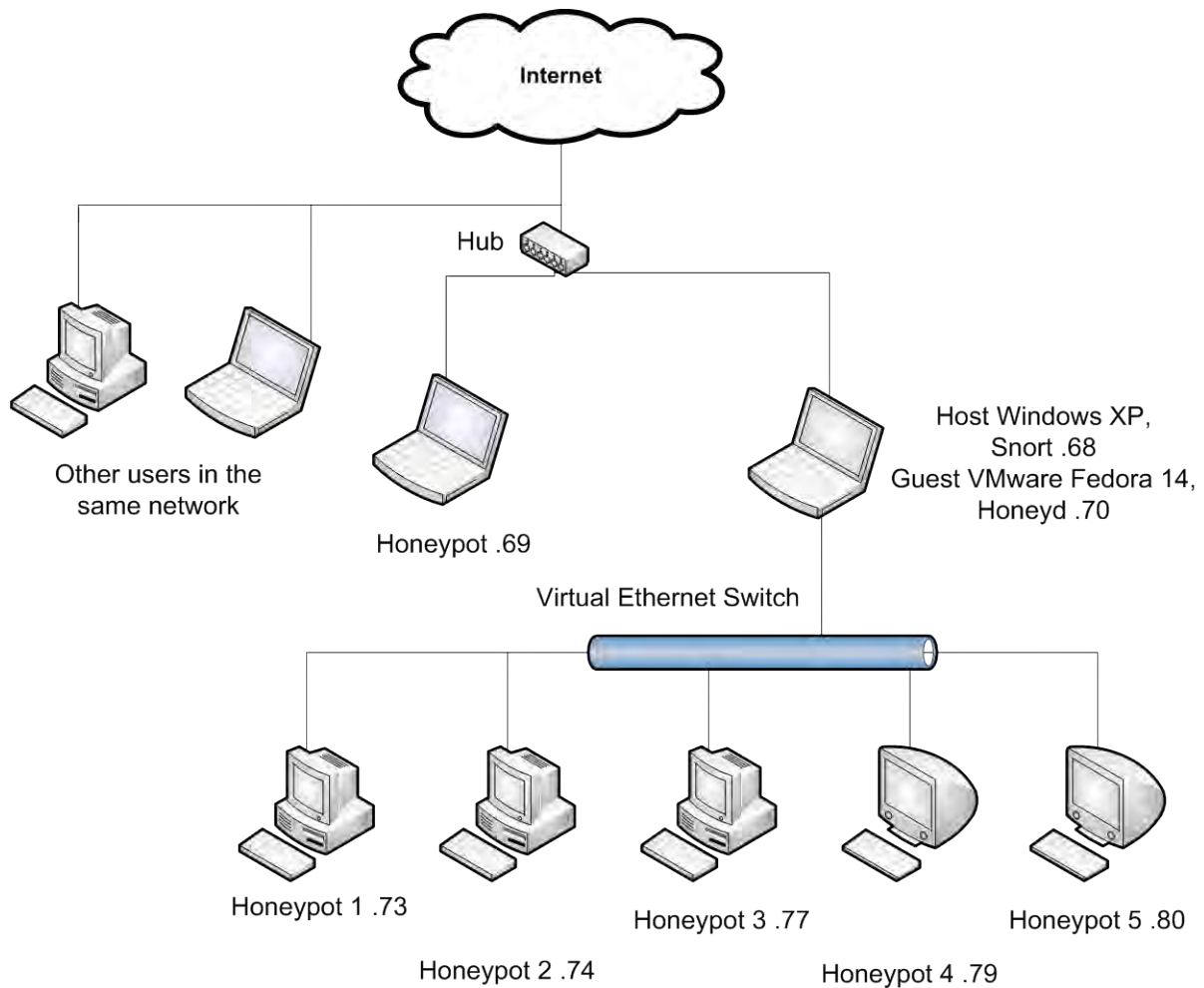
Figure 1.   Network architecture.

C.   SUMMARY OF CONFIGURATIONS USED

In <u>week 1</u> we ran only the host computer with the Snort IDS to have a baseline for the normal traffic of the monitored network.

In <u>week 2</u> we ran Honeyd in the guest virtual machine, mistakenly by default, for a long weekend. This meant that Honeyd ran claiming all the 32 addresses of the network. (Honeyd is much more aggressive than similar programs like LaBrea Tarpit in capturing IP addresses.) We noticed a great increase in the amount of traffic and alerts until the Ethernet connection was closed by the NPS Information Technology and Communications Services department because the experiment was producing IP address conflicts with the valid users of the network.

In <u>week 3</u>, we ran Honeyd on the guest virtual machine claiming five addresses: three simulating Windows hosts and two simulating Linux hosts. Every host had a couple of open ports related with the operating system running—for example, NetBios ports (137, 138, 139) open for Windows hosts or to simulate certain services, like port 80 open for HTTP or port 25 open for SMTP.
To the set of virtual machines we added one running the Honeyd and one as the host for VMware.

In <u>week 4</u> we configured a more elaborate deception that included simulated services (some included in the software Honeyd and others downloaded from the Web page). In addition, we kept most of the open ports and used a more credible ports status.
Two Windows OS computers were simulated and three with Linux OS. Every computer had several TCP and UDP open ports and some emulated services like SMTP, HTTP, FTP, SSH, Telnet,

NetBIOS, POP3, and IMAP. We also used the proxy function of Honeyd in a couple of ports to redirect the attacks to its source.

In week 5, we used a configuration similar to the previous week but without the presence of the considered "production" host.

In week 6, after noticing a decrease in the traffic and number of alerts of the previous configuration, we made some changes to it. Thinking that in some way attackers could be deceived, the IP address of the VM host was switched with that of one of the honeypots. This machine was changed to a Security Onion suite instead of Fedora 14 because it was supposed to have better monitoring capabilities and be hardened against possible attacks. The rest of the configuration was similar to the previous week.

In week 7, although the number of alerts did not increase significantly, we continued with a similar configuration with respect to the previous week, with several scripts running on each host.

In week 8, due to the clear difference in the amount of interactions with emulated Windows and Linux operating systems, we emulated only the Microsoft Windows OS. Analysis of the results so far showed us that not-credible emulated services are probably worse than simulated open ports; as a consequence, we tried again with only open ports and no services running.

In week 9, considering the good results obtained in week 8, we continued with a similar port configuration using four of what appeared to be the most successful scripts.

In <u>week 10</u> we ran the experiment without Honeyd to check again the normal behavior of the network.

In <u>week 11</u> we ran the experiment without Honeyd and even without the guest virtual machine (like week 1, with better tools for analysis), to check again the traffic in the network.

In <u>week 12</u> we ran Honeyd with the same configuration as week 9.

In <u>week 13</u> we continued the same previous configuration, adding a Perl script with a fake Telnet server in port 23 on host .79, and using the newly created personalities for Windows Server 2008 and Windows XP Service Pack 3.

In <u>week 14</u> we tried the last control run without Honeyd running.

In <u>week 15</u> we used the same configuration as week 13, but we replaced the fake Telnet server with a fake internal Web server in maintenance status and we included the proxy function in port 445 in one honeypot, in order to send back to the source every packet the virtual host receives in this port. Also, we switched the IP addresses of four honeypots.

A detailed configuration for every week is available in Appendix A.

D.   METHODOLOGY TO ANALYZE THE RESULTS

Every week, we made a quick analysis of all the information available, using some programs and tools to assist us. At the end of the study, we made a more detailed review.

As we learned what worked and what did not, we used different logs, scripts, tools, and software to better analyze the information captured. This approach required some changes in the methodology and log formats, and as a result, there was a significant difference in the amount of work and information available between the first and last weeks.

We noticed that some of the default formats of the logs are not easy to order or parse for analysis, such as the text alert logs created by Snort. Therefore, we chose the comma-separated values (CSV) format for Snort alerts.

Every week we collected the following logs: Snort summary (displayed on the screen) in a text file, Snort alerts in CSV format, Snort alerts in PCAP format, Honeyd packet logging in text format, and Honeyd system logging in text format.

To analyze Snort alerts, we used Microsoft Log Parser 2.2 in conjunction with scripts in SQL language and HTML templates, to display the alerts in a more friendly way using Web pages. We used some code samples from Giuseppini [5]. For Honeyd logs we also used Microsoft Log Parser 2.

We created "analysis.bat," a small batch program using Log Parser which creates a CSV file with the column headers included. It generates several files and folders by calling six SQL query scripts that produce results in HTML template format. The results of the queries are:

Alerts index: an HTML file that counts the different alerts and displays them in descending order of count.

Alerts details: a folder with an HTML file for every different type of Snort alert. Each file displays in HTML format the information related to the corresponding alert.

Source index: an HTML file that lists the different source IP addresses and displays them in descending order of count.

Source details: a folder with an HTML file for each source IP address related to a Snort alert. Each file displays in HTML format the information related to the corresponding source IP address.

Destination index: an HTML file that lists the different destination IP addresses and displays them in descending order of count.

Destination details: a folder with an HTML file for each destination IP address related to a Snort alert. Each file displays in HTML format the information related to the corresponding destination IP address.

We also created "graph_analysis.bat," a small batch program for Log Parser that generates graphs for several kinds of information: top alerts, top source IPs, top destination IP, alerts per hour, top source port, top destination port, and top protocol.

To analyze Honeyd logs, we create "honeyd_log_analysis.bat," another script in Log Parser. It parses the text file "honeyd.log" related to Honeyd packet logging, generating a CSV file that sets the column name headers to view and process using Excel. This file was useful because it allowed us to see how relatively effective the honeypots were. This was in relation to the interactions they had with other IP addresses, not necessarily alerts or attacks—in other words, the amount of traffic they could attract. With this method, we could quickly compare different honeypot configurations.
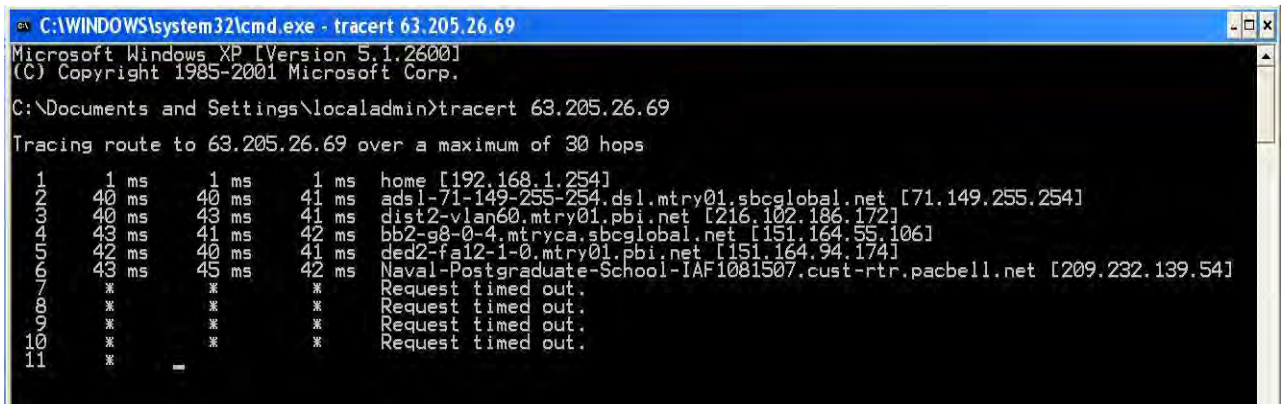
The code and a sample of the results are included as appendices.

With the help of these tools and programs we obtained several statistical values related to the traffic, number of alerts, type of alerts, protocols, and relevant times, all of which gave us much useful information.

V. ANALYSIS OF THE RESULTS

A. THE EXPERIMENT VIEWED FROM THE OUTSIDE

If an attacker scans the network that we set up for this experiment from the outside it is easily identifiable as a government or education resource. Executing a traceroute to any host in the network would show that the name of the last router is clearly associated with the Naval Postgraduate School (see Figure 2). This situation could either deter hackers or increase their interest. But the transparent routeraddress should be easy to fix in other deployments.

Figure 2.   Execution of traceroute from the outside on one IP address of the network

B. HONEYD AS A HONEYPOT

After analysis of the data obtained in weeks 1, 10, 11, and 14, without Honeyd running, we estimated the baseline behavior of the network—the normal network levels of traffic and alerts. Table 3 shows the results.

	week 1	week 10	week 11	week 14
Number of packets	438661	618723	541740	518659
Number of alerts	388	1325	756	488
Different alerts	4	13	16	5
ICMP alerts	388	757	476	488
TCP alerts	0	568	270	0
UDP alerts	0	0	10	0

Table 3.   Statistics of alerts in weeks without Honeyd running

Tables 4 and 5 show corresponding data for weeks when Honeyd was running.

	week 3	week 4	week 5	week 6	week 7
Number of packets	1191410	1313693	701771	906893	740769
Number of alerts	8589	259776	2525	2823	6686
Different alerts	24	36	12	17	11
ICMP alerts	8366	255744	1940	2176	2990
TCP alerts	218	4016	584	647	3696
UDP alerts	5	16	1	0	0

Table 4.   Statistics of alerts in weeks 3–7 with Honeyd running

	week 8	week 9	week 12	week 13	week 15
Number of packets	897552	951556	995235	807712	1066743
Number of alerts	3386	2957	2526	3711	4694
Different alerts	14	19	10	15	14
ICMP alerts	2144	2651	2270	3445	3082
TCP alerts	1242	306	256	266	1612
UDP alerts	0	0	0	0	0

Table 5.   Statistics of alerts in weeks 8–15 with Honeyd running

Comparing these two tables, we see a significant increase (approximately 40%) in the number of packets in the network when Honeyd was running. Also, the total number of alerts increased several times and the number of different alerts was greater. Given these statistics, we can say that Honeyd increases both the amount of traffic and the amount of malicious traffic on a network, confirming that the software is useful for research purposes.

Another way to analyze how successful Honeyd and its different configurations are at changing attacker behavior is by comparing the number of conversations—the traffic between two specific endpoints—produced only by Honeyd, as obtained from its packet log (see Table 6). We can see that with the exception of week 6, the configurations generated significant traffic, with between 40 and 50 thousands interactions for weeks 4, 9, 12 and 15.

Week	Number of interactions
week 4	46124
week 5	14078
week 6	5226
week 7	13676
week 8	26827
week 9	49818
week 12	40950
week 13	35244
week 15	41118

Table 6.   Number of Honeyd interactions per week

A more detailed view of the degree of success of each weekly configuration was obtained from the packet log file (Table 7). We can also see that some honeypots had significantly more conversations than others. In this case, honeypots .77 and .80 had several times more interactions than honeypots .74 and .79.

	week 4 TCP UDP	week 4 ICMP
Honeypot .73	4525	99
Honeypot .74	1720	121
Honeypot .77	18945	371
Honeypot .79	1725	94
Honeypot .80	18283	188

Table 7.   Number of Honeyd interactions by honeypots in week 4

On the contrary, in week 6 we can see that some honeypots were definitively not successful. We can see in Table 8 that three honeypots were completely unsuccessful and only two showed significant interactions.

	week 6 TCP UDP	week 6 ICMP
Honeypot .70	0	0
Honeypot .73	0	4
Honeypot .74	1247	68
Honeypot .77	0	0
Honeypot .79	3755	118

Table 8.   Number of Honeyd interactions by honeypots in week 6

C.   SNORT ALERTS

During the experiment the signature-based network intrusion detection system Snort, running in the host OS, generated alerts for the attacks received in the network. The number and diversity of alerts changed every week; moreover even with the same configuration there were significant changes. A summary of the ten most common alerts is in Table 9.

Alerts	Total
NetBIOS SMB-DS repeated logon failure	7996
Shellcode NOOP	2388
P2P BitTorrent transfer	2364
POLICY remote desktop protocol attempted administrator conn. request	1259
Specific threats ASN.1 constructed bit string	603
NetBIOS SMB-DS Unicode max param overflow attempt	84
NetBIOS DCERPC canonicalize overflow attempt	81
NetBIOS DCERPC remote create instance attempt	27
Web-client Portable executable binary transfer	17
Web-client obfuscated Javascript excessive from CharCode	10

Table 9.   Summary of top 10 alerts in the experiment

The most common alert—plus three others included in the top 10—was related to NetBIOS. These alerts appeared in bursts and only in half of the weeks the experiments were running. An important alert that appeared almost every week was the Shellcode NOOP. This alert was associated with several different Snort signature identification numbers, which means different types of buffer overflow attacks were injecting null operations. Other alerts meriting mention were the attempts to connect to the honeypots and other hosts with remote desktop protocol and the constructed bit string heap corruption.

D.   PORT USAGE

To create a useful honeypot configuration, both for research and decoy purposes, it is necessary to manage the ports. During our experiment, the ports <1024 that were most probed and attacked were in decreasing order of frequency 445, 80, 135, 139, 53 and 22. Port 445, Microsoft-DS (Directory Services) Active Directory, was by far the port most attacked during the experiment with 95.32 % of the TCP protocol alerts. In a distant second place appears port 80, the Hypertext Transfer Protocol (HTTP) port with 3.25% of the same kind of alerts. Also appearing were port 135, Microsoft Endpoint Mapper (also known as DCE/RPC locator service) with 0.84%, and port 139, NetBIOS

Session Service with 0.34%. Thus it is strongly recommended that port 445 be open in any Windows honeypot configuration, along with ports 80, 135, and 139.

E.   OPERATING SYSTEMS MORE ATTACKED

Given that the operating systems emulated by Honeyd were not detected correctly by fingerprinting tools like Nmap (see Appendix C for details), we cannot say that emulated Windows hosts received more attacks than emulated Linux hosts because of their operating system. The main reason why emulated Windows operating systems received more attacks than Linux is that the port most attacked, 445, is a Windows operating-system service.

F.   SERVICE SCRIPTS

One of the most interesting features of Honeyd is its capability to run scripts associated with ports as specified in the Honeyd configuration file. These scripts can improve a decoy to keep the attacker busy interacting with the honeypot for a longer period of time. The degree of success of our scripts was diverse due to two factors. First, as we can see in the high number of alerts in week 4, there was some effect related to the appearance of "new" services (something similar occurred with "new" hosts) that were quickly probed and scanned. Within a short time, between one and two weeks depending on the script, this novelty effect was lost—as is evident in weeks 5 and 6. We can conclude that a script will have a decreasing degree of success if it is repeated for several weeks.

Secondly, services that were not credible because they were old, had errors, or were not logical according to the host's configuration, were quickly recognized by attackers who then avoided the honeypot. We can see this happening in week 6, where the same configuration had been running for three weeks. This identification process took less time as compared with the novelty effect. Hence, for research and decoy purposes, it was better to keep the port open instead of running a non-credible script—as in weeks 8 and 9 (weeks with a small number of scripts but many ports open).

A weakness that also has to be considered is that service scripts could make the operating system running Honeyd more vulnerable to attacks.

G.   POSSIBLE COMPROMISE IN THE SYSTEMS RUNNING THE HONEYPOTS

During the experiment we did not find evidence of compromise of the systems running the honeypots, either in the host OS or the guest OS. The routine update and patching done to the operating systems and antivirus signatures on these hosts probably helped.

H.   HONEYD AS A DECOY

To analyze the usefulness of the Honeyd framework as a decoy in a production environment, we can compare the alerts in production hosts (normal users of the network) with the alerts in the honeypots. This includes the Windows XP running Snort and VMware, and the virtual machine running Honeyd. Our goal was to find evidence that suggests that a significant number of alerts have migrated from production hosts to members of the honeynet.

To do this, we checked the statistics of weeks without Honeyd running: In week 1, 56% of the alerts were on production hosts and 44% in the Windows XP host honeypot. In week 10, 80.7% of the alerts were on production hosts and 19.3% in honeypots. In week 11, 89.5% were on production systems and only 10.5% were on the only honeypot running, the Windows XP host.

To do the same analysis for the weeks where Honeyd was running, we have to discard week 2 because, as a consequence of an error in configuration, we cannot distinguish between the production host and the honeypots in a reliable manner during that week.

In Tables 10 and 11 we can see a significant decrease in the percentage of alerts on production hosts with Honeyd running. There is a notable exception in week 4, which had the highest number of alerts for any week and also the highest number of different alerts, which undoubtedly affected the statistics. Week 4 also had the highest percentage of alerts generated in the production

hosts. How can we explain these contradictory values? As discussed earlier, a configuration with many ports open and dozens of service scripts running was very interesting for attackers; consequently, it was repeatedly probed and attacked. While this situation is good for research purposes, a very tempting network would attract hundreds of attackers that Honeyd as a decoy cannot deceive appropriately. This is true at least with a small number of honeypots, since the production hosts, in practice, will receive much more attacks than without these decoys. In other words, a good configuration as honeypot would not be good as a decoy.

Percentage of alerts	week 3	week 4	week 5	week 6	week 7
Production hosts	4.37769	92.6552	10.1782	9.28091	4.816
Honeypots	95.6223	7.34479	89.8218	90.7191	95.184

Table 10.   Percentage of alerts in production hosts and honeypots with Honeyd running in weeks 1–7

Percentage of alerts	week 8	week 9	week 12	week 13	week 15
Production hosts	11.636	15.996	9.18448	7.78766	5.66681
Honeypots	88.364	84.004	90.8155	92.2123	94.3332

Table 11.   Percentage of alerts in production hosts and honeypots with Honeyd running in weeks 8–15

The rest of the weeks appeared promising for decoy purposes, showing us that more than 80% of the alerts occurred on honeypots instead of production systems. Tables 12 and 13 provide a more detailed analysis of Honeyd. It shows us the percentage of alerts in dividing honeypots into three categories: the host

running Windows XP and VMware, the guest running Fedora 14 or Security Onion, and the honeypots created by Honeyd. We can see that in weeks like 3 and 6, with a high percentage of alerts in honeypots, they were concentrated in the host and guest—a situation not completely desirable because these machines are more vulnerable than the virtual honeypots created by Honeyd. Discarding week 4, the rest of the weeks show that more than 40% of the alerts were on Honeyd virtual honeypots, with numbers as high as 62.8% in week 9, the best for Honeyd as a decoy.

Percentage of alerts	week 3	week 4	week 5	week 6	week 7
Host	72.7209	4.30371	9.82178	12.4336	4.2178
Guest	16.8006	1.48782	38.8911	56.3585	48.564
Honeyd honeypots	**6.10083**	**1.55326**	**41.1089**	**21.927**	**42.402**
Production hosts	4.37769	92.6552	10.1782	9.28091	4.816

Table 12.   Detailed percentage of alerts in production hosts and honeypots with Honeyd running in weeks 3–7

Percentage of alerts	week 8	week 9	week 12	week 13	week 15
Host	7.2652	10.213	10.6097	8.40744	4.55901
Guest	30.892	10.991	26.7221	40.7976	49.8722
Honeyd honeypots	**50.207**	**62.8**	**53.4838**	**43.0073**	**39.902**
Production hosts	11.636	15.996	9.18448	7.78766	5.66681

Table 13.   Detailed percentage of alerts in production hosts and honeypots with Honeyd running in weeks 8–15

THIS PAGE INTENTIONALLY LEFT BLANK

VI. CONCLUSIONS AND FUTURE WORK

A. CONCLUSIONS

Honeyd showed itself to be a useful tool for research on attacks. It increased the amount of traffic in the network by approximately 40%, and by several times the number of alerts available for study.

Honeyd effectively simulated the status of the ports (open, closed, or filtered) and appeared to associate them properly to scripts. However, Honeyd failed to create virtual hosts with a credible emulated operating system because the program was designed for the first-generation Nmap software and has not been updated. The attempts to mitigate this problem were only partially successful; therefore, we had to rely on only the port configuration to simulate different operating systems.

Some of the scripts used were successful to attract many different attackers, but only for a short time. Others were not so successful or lost their initial success quickly. We found two probable reasons for this behavior: The scripts were either not elaborated enough and were easily discovered as deceptions by sophisticated attackers, or Honeyd, as a low-interaction honeypot, has a clear limit in producing credible interactions with attackers.

The ports most attacked with Honeyd were 445 (Microsoft-DS), 80 (HTTP), 135, and 139. This suggests that these ports should be supported in any honeypot configuration for an adequate level of interaction with attackers. Ports 445, 135, and 139 are related to Windows operating systems, and hence produced more attacks for virtual hosts simulating Windows than the Linux operating system.

The most common attacks received in the honeypots were related to NetBIOS and to Shellcode NOOPs (buffer overflows). Other significant attacks were the attempts to connect to the honeypots and other hosts with a remote-desktop protocol and the constructed bit string heap corruption. However, we did

not find symptoms or evidence of successful attacks or compromise in the host running Windows XP, nor did we find it in the guest system running Fedora 14 or Security Onion Ubuntu. In general, we found that the honeypots received scans and basic attacks, without showing that advanced attack techniques were used.

The use of the software Microsoft Log Parser with sql and tlp scripts appeared to be a useful and flexible tool for analyzing Snort alerts in Windows machines.

Honeyd appeared also to be useful as a decoy to distract attackers from more valuable targets, but it must be carefully configured to achieve good results. Some configurations with many open ports and scripts running were useful as a research honeypot, but not as a decoy. Good configurations for decoys must have a limited number of scripts and open ports, enough to make the decoys attractive, but not so many as to make the entire network more attractive to attackers. When this situation happened, the normal users of the network—i.e., the hosts that were intended to be protected by the decoys—were attacked considerably more than usual. But decoys, as in conventional warfare, have a limited useful life. They must be used at the right time.

B.   FUTURE WORK

We suggest the following future investigations:

- Modify Honeyd to allow second-generation operating-system detection (this could be done by modifying the file "personality.c") and run the experiment again.

- Run the experiment with more IP addresses available, so Honeyd can simulate more virtual hosts as honeypots.

- Modify or create scripts that can deceive attackers for a longer period of time. We suggest elaborated fake Web servers, with a credible degree of interaction during the navigation on the Web site. This should be complemented with messages about restrictions to navigate in some pages, login windows, and banners, encouraging

36

unauthorized users to close the page. Mail servers and file-transfer servers could also be simulated.

- Try a similar experiment with other software, such as Nepenthes or its successor Dionaea, to deploy low-interaction honeypots.

- Experiment with high-interaction honeypots based in virtualization software like VMware or User-mode Linux (UML).

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX A.   DETAILS OF THE CONFIGURATIONS USED BY WEEK

Week 1:

We ran the laptop with the host operating system (Windows XP) in IP address
.68 with the Snort IDS running, to get a baseline of the normal traffic and alerts of
the monitored network.

Week 2:

We ran Honeyd for a long weekend, claiming (by mistake) all the 32 addresses of
the network. Honeyd is more aggressive than similar programs like LaBrea Tarpit
in claiming IP addresses.

Week 3:

We ran Honeyd in the guest operating system (Fedora 14), claiming 5 IP
addresses:

- .73 "Microsoft Windows 98 SP2" with TCP ports 135,137,139, and 9898
  open.
- .74 "Microsoft Windows XP Professional SP1" with TCP ports 135, 137,
  139 and 443 open.
- .77 "Microsoft Windows NT 4.0 Server SP5-SP6" with TCP ports 80, 137,
  and 139 open.
- .79 "OpenBSD 3.3" with TCP and UDP port 421 open.
- .80 "FreeBSD 4.7-RELEASE" with TCP and UDP port 4448 open. Default
  TCP action tarpit open.

To these five virtual machines we had to add the .70 guest operating system
running the Honeyd and .68 as the host operating system for VMware. The latter
is the only real machine used in the experiment.

Week 4:

We made a more elaborate configuration for the same IP addresses that included simulated services (some included in the software Honeyd and others downloaded from the Honeyd Web page) and more credible ports status.

- .73 "Microsoft Windows NT 4.0 Server SP5-SP6" with TCP ports 21, 25, 80, 110, 137, 138, 139, 143, 389, 445, and 5901, and UDP ports 161, 137, 138, and 445 open and running simulated services using scripts.
- .74 "Linux 2.4.7 (X86)" with TCP ports 21, 22, 23, 25, 79, 80, 110, 143, 515, 3128 , 8080, and 8081, and UDP ports 53, 161, and 514 open and running simulated services using scripts
- .77 "Microsoft Windows NT 4.0 Server SP5-SP6" with the same configuration as week 3 but with a script on TCP port 80 and TCP port 8080 open.
- .79 "Linux 2.2.12 - 2.2.19" with TCP ports 22, 23, 25, 79, 80, 110, 143, 515, 3128 , 8080, and 8081, and UDP ports 53, 161, and 514 open and running simulated services using scripts.
- .80 "FreeBSD 4.7-RELEASE" using the same configuration as in week 3.

Week 5:

This week the configuration was the same as the previous week.

Week 6:

In this week we made some important changes to the configuration of the experiment: We changed the guest operating system to .80 instead of the previously used IP address .70. As a consequence, the honeypot with FreeBSD emulated operating system was moved to IP address .70. The VM guest operating system was changed to a Security Onion suite instead of Fedora 14 because it is supposed to have better capabilities for monitoring and be hardened against possible attacks.

- .70 "FreeBSD 4.7-RELEASE" using the same previous configuration for this operating system.
- .73 "Microsoft Windows NT 4.0 Server SP5-SP6" with TCP ports 21, 25, 80, 110, 137, 138, 139, 143, 389, 445, and 5901, and UDP ports 161, 137, 138, and 445 open and running simulated services using scripts.
- .74 "Linux 2.4.7 (X86)" with TCP ports 21, 22, 23, 25, 79, 80, 110, 143, 515, 3128, 8080, and 8081, and UDP ports 53, 161, and 514 open and running simulated services using scripts
- .77 "Microsoft Windows NT 4.0 Server SP5-SP6" with the same configuration as week 3 but with a script on TCP port 80 and TCP port 8080 open.
- .79 "Linux 2.2.12 - 2.2.19" with TCP ports 22, 23, 25, 79, 80, 110, 143, 515, 3128 , 8080, and 8081, and UDP ports 53, 161, and 514 open and running simulated services using scripts.
- .80 Security Onion guest operating system running honeyd.

Week 7:

On week 7 the configuration was the following:
- .70 "Linux 2.4.7 (X86)" with TCP ports 21, 22, 23, 25, 79, 80, 110, 143, 515, 3128, 8080, 8081, and UDP ports 53, 161, and 514 open and running simulated services using scripts
- .73 "Microsoft Windows NT 4.0 Server SP5-SP6" with TCP port 80 (HTTP) running a service script; TCP ports 137, 139, 443, and 8080 open; and UDP ports 135 and 137.
- .74 "Linux 2.4.7 (X86)" with TCP port 23 (Telnet) and TCP port 79 (Name/finger protocol) with scripts running and the UDP port open.
- .77 the same as .70
- .79 "Microsoft Windows NT 4.0 Server SP5-SP6" with ports TCP 21, 25, 80, 110, 137, 138, 139, 143, 389, 445, and 5901, and UDP ports 161, 137, 138, and 445 open and running simulated services using scripts.

<u>Week 8:</u>

In this week, due to the clear difference in the number of interactions between the Windows and Linux operating systems, we configured only Windows. Analysis of the results thus far showed us that not-credible emulated services are probably worse than simulated only-open ports; as a consequence, we tried again with open ports instead of simulated services.

The configuration was the following:

- .70 "Microsoft Windows XP Professional SP1" with TCP ports 135, 139, and 445 open, and UDP ports 135, 137, 138, 445, and 4500 open.
- .73 "Microsoft Windows Server 2003 Standard Edition" with TCP ports 20, 21, 25, 80, 135, 139, and 445 open, and UDP ports 53, 135, 137, 138, and 445 open.
- .74 "Microsoft Windows XP Professional SP1" with TCP ports 135, 139, and 445 open, and UDP ports 135, 137, 138, 445, and 4500 open.
- .77 "Microsoft Windows NT 4.0 Server SP5-SP6" with TCP ports 80, 135, 139, 443, 445, and 8080 open, and UDP ports 135, 137, 138, and 445 open.
- .79 "Microsoft Windows Server 2003 Standard Edition" with TCP ports 135, 139, 445, and 1433 open, and UDP ports 135, 137, 138, 445, and 1434 open.

<u>Week 9:</u>

On week 9 the configuration was the following:

- .70 "Microsoft Windows XP Professional SP1" with TCP ports 135, 139, and 445 open, and UDP ports 135, 137, 138, 445, and 4500 open.
- .73 "Microsoft Windows Server 2003 Standard Edition" with TCP ports 20, 21, 25, 80, 135, 139, and 445 open, and UDP ports 53, 135, 137, 138, and 445 open. Scripts were running in ports 21 (FTP), 25 (SMTP), and 110 (POP3).

- .74 "Microsoft Windows XP Professional SP1" with TCP ports 135, 139, and 445 open, and UDP ports 135, 137, 138, 445, and 4500 open.
- .77 "Microsoft Windows NT 4.0 Server SP5-SP6 with TCP ports 80, 135, 139, 445, and 8080 open, and UDP ports 135, 137, 138, and 445. Perl script running in TCP port 80 (HTTP).
- .79 "Microsoft Windows Server 2003 Standard Edition" with TCP ports 135, 139, 445, and 1433 open, and UDP ports 135, 137, 138, 445, and 1434 open.

Week 10:

We ran the experiment without Honeyd to check the normal behavior of the network in a different time period.

Week 11:

We ran the experiment without Honeyd, this time even without the guest virtual machine (as in week 1, although with better tools for analysis), to check again the traffic in the network.

Week 12:

Due to the previous week's promising results, we repeated the same configuration as week 9.

Week 13:

We continued the same previous week configuration, adding a Perl script with a fake Telnet server in port 23 on host .79, and using the newly created personalities for Windows Server 2008 and Windows XP Service Pack 3.

Week 14:

We tried the last control run without Honeyd running.

We used the same configuration as week 13, but we replaced the fake Telnet server with a fake internal web server in maintenance status, and we included the proxy function in port 445 in one honeypot (.77), in order to send back to the source every packet the virtual host receives in this port. We also switched the IP addresses of 4 honeypots.

APPENDIX B.   COMMANDS, CONFIGURATION, AND CODE USED

A.   COMMANDS USED

Snort was run in a Windows XP laptop with these instructions in the command line:

Snort.exe –dev –i2 –y –c C:\Snort\etc\snort.conf –l C:\Snort\log

-dev: Tells Snort to display the packet data, decode data link layer headers and be verbose. Sometimes this switch was not used.

-i2: Tells Snort which interface has to be monitored.

-y: Tells Snort to include the year in the timestamp format. It is necessary to be compatible with Microsoft Log Parser.

-c: Designates the configuration file and location that Snort uses.

-l: Creates a log file in the indicated location and according to the format selected in the configuration file.

Honeyd was run in a Fedora 14 VM and a Security Onion VM with this command in the terminal:

honeyd –d –f /home/user/Desktop/honeyd.conf –l
/home/user/Desktop/honeyd.log –s /home/user/Desktop/honeyd_syslog.log

-d: This switch tells Honeyd not to daemonize and display verbose messages of the activities that it is doing.

-f: Specifies the configuration file that Honeyd uses.

-l: Creates packet-level log file in the indicated location.

-s: Creates service-level log file in the indicated location.

B. HONEYD CONFIGURATION FILE

This file is one of the most important files of Honeyd. On it we define the network configuration, the emulated operating systems, the status of ports and the services running.

For example, the configuration file for Honeyd during week 9 was the following:

```
### Configuration for week 9

route entry 63.205.26.65

route 63.205.26.65 link 63.205.26.70/32
route 63.205.26.65 link 63.205.26.73/32
route 63.205.26.65 link 63.205.26.74/32
route 63.205.26.65 link 63.205.26.77/32
route 63.205.26.65 link 63.205.26.79/32

### Windows NT4 web server
create windows
set windows personality "Microsoft Windows NT 4.0 Server SP5-SP6"
add           windows          tcp          port          80          "perl
/home/erwin/Desktop/honeyd_services_scripts/iisemulator-0.95/iisemul8.pl"
add windows udp port 135 open
add windows tcp port 135 open
add windows udp port 137 open
add windows udp port 138 open
add windows tcp port 139 open
add windows tcp port 443 open
add windows udp port 445 open
add windows tcp port 445 open
add windows tcp port 8080 open
set windows default tcp action reset
set windows default udp action reset
set windows uptime 168939
set windows droprate in 4

### Windows SQL Server
create windowsSQL
set windowsSQL personality "Microsoft Windows Server 2003 Standard Edition"
add windowsSQL tcp port 135 open
add windowsSQL udp port 135 open
add windowsSQL udp port 137 open
add windowsSQL udp port 138 open
add windowsSQL tcp port 139 open
```

add windowsSQL tcp port 445 open
add windowsSQL udp port 445 open
add windowsSQL udp port 1434 open
add windowsSQL tcp port 1433 open
set windowsSQL default tcp action reset
set windowsSQL default udp action reset
set windowsSQL ethernet "intel"

Windows 2003 Server
create windows2003
set windows2003 personality "Microsoft Windows Server 2003 Standard Edition"
add windows2003 tcp port 20 open
add   windows2003   tcp   port   21   "sh
/home/erwin/Desktop/honeyd_services_scripts/ftp.sh"
add   windows2003   tcp   port   25   "sh
/home/erwin/Desktop/honeyd_services_scripts/smtp.sh"
add windows2003 udp port 53 open
add windows2003 tcp port 80 open
add   windows2003   tcp   port   110   "sh
/home/erwin/Desktop/honeyd_services_scripts/pop3.sh"
add windows2003 udp port 110 open
add windows2003 tcp port 135 open
add windows2003 udp port 135 open
add windows2003 udp port 137 open
add windows2003 udp port 138 open
add windows2003 tcp port 139 open
add windows2003 tcp port 445 open
add windows2003 udp port 445 open
set windows2003 default tcp action reset
set windows2003 default udp action reset
set windows2003 uptime 147239
set windows2003 droprate in 8
set windows2003 ethernet "00:24:E8:A3:d2:f1"

Windows XP
create windowsXP
set windowsXP personality "Microsoft Windows XP Professional SP1"
add windowsXP tcp port 135 open
add windowsXP udp port 135 open
add windowsXP udp port 137 open
add windowsXP udp port 138 open
add windowsXP tcp port 139 open
add windowsXP tcp port 445 open
add windowsXP udp port 445 open
add windowsXP udp port 4500 open

```
set windowsXP default tcp action reset
set windowsXP default udp action reset
set windowsXP ethernet "00:24:E8:23:d0:4f"


bind 63.205.26.70 windowsXP
bind 63.205.26.73 windows2003
bind 63.205.26.74 windowsXP
bind 63.205.26.77 windows
bind 63.205.26.79 windowsSQL
```

This file specifies for Honeyd: the default gateway (route entry), the IP addresses available to create virtual hosts, several different hosts with particular characteristics in relation to its emulated operating system, TCP and UDP ports open, ports with services scripts running, default action in other ports, time the host is up, packet drop rate, MAC addresses and which IP address is assigned to each host.

C. SCRIPTS AND CODE USED

Figure 3 shows how the data was processed and analyzed.

Alerts.csv
(from Snort)

Honeyd.log
(from
Honeyd)

analysis.bat

graph_analysis.bat

honeyd_log_analysis.bat

Alertheader.csv
Alerts-index.sql
Alerts-index.tpl
Alerts-Details.sql
Alerts-Details.tpl
Srcip-details.sql
Srcip-details.tpl
Dstip-details.sql
Dstip-details.tpl

GraphAlertsPerHour.sql
GraphTopAlerts.sql
GraphTopDstIPs.sql
GraphTopDstPorts.sql
GraphTopProtocol.sql
GraphTopSrcIPs.sql
GraphTopSrcPorts.sql

Honeydlog_header.tsv

honeyd1.csv

alerts.html
alerts folder
srcip.html
srcip folder
dstip.html
dstip folder
alertfin.csv

AlertsbyHour.gif
AlertsTopAlerts.gif
AlertsTopDstIPs.gif
AlertsTopDstPorts.gif
AlertsTopProtocols.gif
AlertsTopSrcIPs.gif
AlertsTopSrcPorts.gif

Figure 3.   Flow diagram of the scripts and programs used to analyze the results
every week

<u>Analysis.bat:</u>

LogParser -i:CSV -o:CSV -headerRow:off -iTsFormat:MM/dd/yy-hh:mm:ss -iHeaderFile:AlertHeader.csv "SELECT* INTO report/alertfin1.csv FROM alert.csv"

LogParser file:alerts_index.sql -i:CSV -iHeaderFile:AlertHeader.csv -iTsFormat:MM/dd/yy-hh:mm:ss -headerRow:off -o:tpl -tpl:alerts_index.tpl

LogParser file:alerts_detail.sql -i:CSV -iHeaderFile:AlertHeader.csv -iTsFormat:MM/dd/yy-hh:mm:ss -headerRow:off -o:tpl -tpl:alerts_detail.tpl

LogParser file:srcip_index.sql -i:CSV -iHeaderFile:AlertHeader.csv -iTsFormat:MM/dd/yy-hh:mm:ss -headerRow:off -o:tpl -tpl:srcip_index.tpl

LogParser file:srcip_detail.sql -i:CSV -iHeaderFile:AlertHeader.csv -iTsFormat:MM/dd/yy-hh:mm:ss -headerRow:off -o:tpl -tpl:srcip_detail.tpl

LogParser file:dstip_index.sql -i:CSV -iHeaderFile:AlertHeader.csv -iTsFormat:MM/dd/yy-hh:mm:ss -headerRow:off -o:tpl -tpl:dstip_index.tpl

LogParser file:dstip_detail.sql -i:CSV -iHeaderFile:AlertHeader.csv -iTsFormat:MM/dd/yy-hh:mm:ss -headerRow:off -o:tpl -tpl:dstip_detail.tpl

<u>AlertHeader.csv</u> (Adapted from Giuseppini, 2005 [5])

timestamp, sig_generator, sig_id, sig_rev, msg, proto, src, srcport, dst, dstport, ethsrc, ethdst, ethlen, tcpflags, tcpseq, tcpack, tcplen, tcpwindow, ttl, tos, id, dgmlen, iplen, icmptype, icmpcode, icmpid, icmpseq

<u>Alerts-Index.sql</u> (From Giuseppini, 2005 [5])

```
SELECT DISTINCT
        sig_id,
        msg,
        COUNT(msg) as Alerts
INTO report\alerts.html
FROM alert.csv
GROUP BY msg, sig_id
ORDER BY Alerts DESC
```


<u>Alerts-Index.tpl</u> (Adapted from Giuseppini, 2005 [5])

```
<LPHEADER>
        <html>
        <head>
                <meta http-equiv="Content-Type" content="text/html; charset=windows-1252">
                <link rel="stylesheet" type="text/css" href="snort.css">
                <title>Snort Alert Messages</title>
```

50

```
</head>
<body>
<p><h1>Snort Alerts Summary</h1><br/>
<i>Created %SYSTEM_TIMESTAMP% </i></p>
        <table border="0" width="75%" cellspacing="2">
        <tr>
                <th><b>Signature</b></th>
                <th><b>Message</b></th>
                <th><b>Alerts</b></th>
        </tr>
</LPHEADER>
<LPBODY>
        <tr>
                <td><a
href=http://www.snort.org/search/> %sig_id%</a></td>
                <td> %msg%</td>
                <td><a href=alert\%sig_id%.html> %Alerts%</a></td>
        </tr>
</LPBODY>
<LPFOOTER>
    </table>
    </p>
    </body>
    </html>
</LPFOOTER>
```

<u>Alerts-Detail.sql</u> (Adapted from Giuseppini, 2005 [5])

```
SELECT
    sig_id,
    TO_DATE(timestamp) AS Date,
    TO_TIME(timestamp) AS Time,
    msg,
    proto,
    src,
    srcport,
    dst,
    dstport,
    ethsrc,
    ethdst,
    ethlen,
```

```
        tcpflags,
        tcpseq,
        tcpack,
        tcplen,
        tcpwindow,
        ttl,
        tos,
        id,
        dgmlen,
        iplen,
        icmptype,
        icmpcode,
        icmpid,
        icmpseq
INTO report\alert\*.html
FROM alert.csv
```

Alerts-Detail.tpl (Adapted from Giuseppini, 2005 [5])

```
<LPHEADER>
                <table border="0" width="140%" cellspacing="2">
                <tr>
                        <th><b>date</b></th>
                        <th><b>time</b></th>
                        <th><b>proto</b></th>
                        <th><b>src</b></th>
                        <th><b>srcport</b></th>
                        <th><b>dst</b></th>
                        <th><b>dstport</b></th>
                        <th><b>ethsrc</b></th>
                        <th><b>ethdst</b></th>
                        <th><b>ethlen</b></th>
                        <th><b>tcpflags</b></th>
                        <th><b>tcpseq</b></th>
                        <th><b>tcpack</b></th>
                        <th><b>tcplen</b></th>
                        <th><b>tcpwindow</b></th>
                        <th><b>ttl</b></th>
                        <th><b>tos</b></th>
                        <th><b>id</b></th>
                        <th><b>dgmlen</b></th>
```

```
                          <th><b>iplen</b></th>
                          <th><b>icmptype</b></th>
                          <th><b>icmpcode</b></th>
                          <th><b>icmpid</b></th>
                          <th><b>icmpseq</b></th>
                  </tr>
</LPHEADER>
<LPBODY>
                  <tr>
                          <td> %date%</td>
                          <td> %time%</td>
                          <td> %proto%</td>
                          <td> <a href=..\src\%src%.html>%src%</a></td>
                          <td> %srcport%</td>
                          <td> <a href=..\dst\%dst%.html>%dst%</a></td>
                          <td> %dstport%</td>
                          <td> %ethsrc%</td>
                          <td> %ethdst%</td>
                          <td> %ethlen%</td>
                          <td> %tcpflags%</td>
                          <td> %tcpseq%</td>
                          <td> %tcpack%</td>
                          <td> %tcplen%</td>
                          <td> %tcpwindow%</td>
                          <td> %ttl%</td>
                          <td> %tos%</td>
                          <td> %id%</td>
                          <td> %dgmlen%</td>
                          <td> %iplen%</td>
                          <td> %icmptype%</td>
                          <td> %icmpcode%</td>
                          <td> %icmpid%</td>
                          <td> %icmpseq%</td>
                  </tr>
</LPBODY>
<LPFOOTER>
        </table>
        </p>
        </body>
        </html>
```

</LPFOOTER>

srcip-detail.sql (Adapted from Giuseppini, 2005 [5])

```sql
SELECT
        src,
        TO_DATE(timestamp) AS Date,
        TO_TIME(timestamp) AS Time,
        msg,
        proto,
        sig_id,
        srcport,
        dst,
        dstport,
        ethsrc,
        ethdst,
        ethlen,
        tcpflags,
        tcpseq,
        tcpack,
        tcplen,
        tcpwindow,
        ttl,
        tos,
        id,
        dgmlen,
        iplen,
        icmptype,
        icmpcode,
        icmpid,
        icmpseq
INTO report\srcip\*.html
FROM alert.csv
```

srcip-detail.tpl (Adapted from Giuseppini, 2005 [5])

```html
<LPHEADER>
                <table border="0" width="140%" cellspacing="2">
                <tr>
                        <th><b>date</b></th>
                        <th><b>time</b></th>
                        <th><b>proto</b></th>
```

```
                        <th><b>sig_id</b></th>
                        <th><b>srcport</b></th>
                        <th><b>dst</b></th>
                        <th><b>dstport</b></th>
                        <th><b>ethsrc</b></th>
                        <th><b>ethdst</b></th>
                        <th><b>ethlen</b></th>
                        <th><b>tcpflags</b></th>
                        <th><b>tcpseq</b></th>
                        <th><b>tcpack</b></th>
                        <th><b>tcplen</b></th>
                        <th><b>tcpwindow</b></th>
                        <th><b>ttl</b></th>
                        <th><b>tos</b></th>
                        <th><b>id</b></th>
                        <th><b>dgmlen</b></th>
                        <th><b>iplen</b></th>
                        <th><b>icmptype</b></th>
                        <th><b>icmpcode</b></th>
                        <th><b>icmpid</b></th>
                        <th><b>icmpseq</b></th>
                </tr>
</LPHEADER>
<LPBODY>
                <tr>
                        <td> %date%</td>
                        <td> %time%</td>
                        <td> %proto%</td>
                        <td> %sig_id%</td>
                        <td> %srcport%</td>
                        <td> %dst%</td>
                        <td> %dstport%</td>
                        <td> %ethsrc%</td>
                        <td> %ethdst%</td>
                        <td> %ethlen%</td>
                        <td> %tcpflags%</td>
                        <td> %tcpseq%</td>
                        <td> %tcpack%</td>
                        <td> %tcplen%</td>
                        <td> %tcpwindow%</td>
```

```
                        <td> %ttl%</td>
                        <td> %tos%</td>
                        <td> %id%</td>
                        <td> %dgmlen%</td>
                        <td> %iplen%</td>
                        <td> %icmptype%</td>
                        <td> %icmpcode%</td>
                        <td> %icmpid%</td>
                        <td> %icmpseq%</td>
                </tr>
</LPBODY>
<LPFOOTER>
        </table>
        </p>
        </body>
        </html>
</LPFOOTER>
```

dstip-detail.sql (Adapted from Giuseppini, 2005 [5])

```
SELECT
        dst,
        TO_DATE(timestamp) AS Date,
        TO_TIME(timestamp) AS Time,
        msg,
        proto,
        sig_id,
        srcport,
        src,
        dstport,
        ethsrc,
        ethdst,
        ethlen,
        tcpflags,
        tcpseq,
        tcpack,
        tcplen,
        tcpwindow,
        ttl,
        tos,
        id,
```

```
        dgmlen,
        iplen,
        icmptype,
        icmpcode,
        icmpid,
        icmpseq
INTO report\dstip\*.html
FROM alert.csv
```

dstip-detail.tpl (Adapted from Giuseppini, 2005 [5])

```
<LPHEADER>
                <table border="0" width="140%" cellspacing="2">
                <tr>
                        <th><b>date</b></th>
                        <th><b>time</b></th>
                        <th><b>proto</b></th>
                        <th><b>src</b></th>
                        <th><b>srcport</b></th>
                        <th><b>sig_id</b></th>
                        <th><b>dstport</b></th>
                        <th><b>ethsrc</b></th>
                        <th><b>ethdst</b></th>
                        <th><b>ethlen</b></th>
                        <th><b>tcpflags</b></th>
                        <th><b>tcpseq</b></th>
                        <th><b>tcpack</b></th>
                        <th><b>tcplen</b></th>
                        <th><b>tcpwindow</b></th>
                        <th><b>ttl</b></th>
                        <th><b>tos</b></th>
                        <th><b>id</b></th>
                        <th><b>dgmlen</b></th>
                        <th><b>iplen</b></th>
                        <th><b>icmptype</b></th>
                        <th><b>icmpcode</b></th>
                        <th><b>icmpid</b></th>
                        <th><b>icmpseq</b></th>
                </tr>
</LPHEADER>
<LPBODY>
```

```
                <tr>
                        <td> %date%</td>
                        <td> %time%</td>
                        <td> %proto%</td>
                        <td> %src%</td>
                        <td> %srcport%</td>
                        <td> %sig_id%</td>
                        <td> %dstport%</td>
                        <td> %ethsrc%</td>
                        <td> %ethdst%</td>
                        <td> %ethlen%</td>
                        <td> %tcpflags%</td>
                        <td> %tcpseq%</td>
                        <td> %tcpack%</td>
                        <td> %tcplen%</td>
                        <td> %tcpwindow%</td>
                        <td> %ttl%</td>
                        <td> %tos%</td>
                        <td> %id%</td>
                        <td> %dgmlen%</td>
                        <td> %iplen%</td>
                        <td> %icmptype%</td>
                        <td> %icmpcode%</td>
                        <td> %icmpid%</td>
                        <td> %icmpseq%</td>
                </tr>
</LPBODY>
<LPFOOTER>
        </table>
        </p>
        </body>
        </html>
</LPFOOTER>
```

Graph_analysis.bat: (Adapted from Giuseppini, 2005 [5])

```
Logparser      file:GraphTopAlerts.sql      -i:csv      -iHeaderFile:AlertHeader.csv      -
iTsFormat:MM/dd/yy-hh:mm:ss      -headerRow:off      -o:chart      -chartType:Pie3D      -
groupSize:1600x800 -values:ON -chartTitle:"Top Alerts" -categories:OFF
```

Logparser   file:GraphTopSrcIPs.sql   -i:csv   -iHeaderFile:AlertHeader.csv   -
iTsFormat:MM/dd/yy-hh:mm:ss   -headerRow:off   -o:chart   -chartType:Pie   -
groupSize:1600x800 -values:OFF -chartTitle:"Top Source IP" -categories:OFF

Logparser   file:GraphAlertsPerHour.sql   -i:csv   -iHeaderFile:AlertHeader.csv   -
iTsFormat:MM/dd/yy-hh:mm:ss   -headerRow:off   -o:chart   -chartType:smoothline   -
groupSize:1400x700 -values:OFF -chartTitle:"Alerts per Hour" -categories:OFF

Logparser   file:GraphTopDstPorts.sql   -i:csv   -iHeaderFile:AlertHeader.csv   -
iTsFormat:mm/dd/yy-hh:mm:ss   -headerRow:off   -o:chart   -chartType:BarStacked   -
groupSize:1200x600 -values:OFF -chartTitle:"Top Destination Ports"

Logparser   file:GraphTopSrcPorts.sql   -i:csv   -iHeaderFile:AlertHeader.csv   -
iTsFormat:MM/dd/yy-hh:mm:ss   -headerRow:off   -o:chart   -chartType:BarStacked   -
groupSize:1200x600 -values:OFF -chartTitle:"Top Source Ports"

Logparser   file:GraphTopDstIPs.sql   -i:csv   -iHeaderFile:AlertHeader.csv   -
iTsFormat:MM/dd/yy-hh:mm:ss   -headerRow:off   -o:chart   -chartType:Pie   -
groupSize:1600x800 -values:OFF -chartTitle:"Top Destination IP" -categories:OFF

Logparser   file:GraphTopProtocols.sql   -i:csv   -iHeaderFile:AlertHeader.csv   -
iTsFormat:MM/dd/yy-hh:mm:ss   -headerRow:off   -o:chart   -chartType:Pie   -
groupSize:1000x500 -values:ON -chartTitle:"Top Protocols" -categories:OFF

GraphTopAlerts.sql
```
SELECT
        msg, ---sig_id,
        Count(msg) as Alerts
INTO report\AlertsTopAlerts.gif
FROM alert.csv
GROUP BY msg ---sig_id
ORDER BY Alerts DESC
```

GraphTopSrcIPs.sql
```
SELECT
        src,
        Count(msg) as Alerts
INTO report\AlertsTopSrcIPs.gif
FROM alert.csv
GROUP BY src
ORDER BY Alerts DESC
```

GraphAlertsPerHour.sql
```
SELECT
        Count(*) as Alerts
USING QUANTIZE(timestamp,300) as Hour
INTO report\AlertsByHour.gif
FROM alert.csv
```

GROUP BY Hour

GraphTopDstPorts.sql

GraphTopSrcPorts.sql
SELECT TOP 10
        STRCAT(STRCAT(TO_STRING(srcport),' - '), proto) AS Source,
        Count(*) as Alerts
        USING src as SourcePort
INTO report\AlertsTopSrcPorts.gif
FROM alert.csv
GROUP BY Source
ORDER BY Alerts DESC

GraphTopDstIPs.sql
SELECT
        dst,
        Count(msg) as Alerts
INTO report\AlertsTopDstIPs.gif
FROM alert.csv
GROUP BY dst
ORDER BY Alerts DESC

GraphTopProtocols.sql
SELECT
        proto, ---sig_id,
        Count(proto) as Alerts
INTO report\AlertsTopProtocols.gif
FROM alert.csv
GROUP BY proto ---sig_id
ORDER BY Alerts DESC

Honeyd_log_analysis.bat:

LogParser   -i:TSV   -o:CSV   -iHeaderFile:honeydlog_header.tsv   -headerRow:off   -
iSeparator:space "SELECT* INTO honeyd1.csv FROM honeyd.log"

honeydlog_header.tsv:

timestamp proto T srcIP srcPort destIP destPort Info Comment

APPENDIX C.   NMAP OS DETECTION AGAINST HONEYD

To specify the parameters used to emulate operating systems at TCP/IP stack level, Honeyd uses the same database "nmap.prints" file that is included in the Nmap software to properly detect specific operating systems. In this way, Honeyd tries to give the responses Nmap is expecting to receive from the probes and packets it sends.

Honeyd's configuration specifies how to respond to as many as nine TCP/IP packets and probes sent by the scanner:

- TSeq (Test sequence) specifies how to derive the TCP packet sequence numbers.
- T1 (Test 1), specifies how to respond to a SYN packet sent to an open TCP port.
- T2, specifies how to respond to a NULL packet sent to an open TCP port.
- T3, specifies how to respond to a SYN, FIN, PSH, and URG packet sent to an open TCP port.
- T4, specifies how to respond to an ACK packet sent to an open TCP port.
- T5, specifies how to respond to a SYN packet sent to a closed TCP port.
- T6, specifies how to respond to an ACK packet sent to a closed TCP port.
- T7, specifies how to respond to a FIN, PSH, and URG packet sent to a closed TCP port.
- PU, specifies how to respond to a probe sent to a closed UDP port.

For example, the following results are expected for these tests for a Windows XP OS updated with Service Pack 1.

Fingerprint Microsoft Windows XP Professional SP1

Class   Microsoft | Windows | NT/2K/XP | general   purpose

TSeq(Class=RI%gcd=<6%SI=<9A6AA&>18A0%IPID=I%TS=U)

T1(DF=Y%W=8820%ACK=S++|O%Flags=AS|A%Ops=M|)

T2(Resp=Y%DF=N%W=0%ACK=S%Flags=AR%Ops=)

T3(Resp=Y%DF=Y%W=8820%ACK=S++|O%Flags=AS||A%Ops=M||)

T4(DF=N%W=0%ACK=O%Flags=R%Ops=)

T5(DF=N%W=0%ACK=S++%Flags=AR%Ops=)

T6(DF=N%W=0%ACK=O%Flags=R%Ops=)

T7(DF=N%W=0%ACK=S++%Flags=AR%Ops=)

PU(DF=N%TOS=0%IPLEN=38%RIPTL=148%RID=E%RIPCK=E%UCK=E%ULEN=134%

DAT=E)

This first-generation operating-system detection method used by earlier versions of Nmap worked well with Honeyd. But on December 2007, a second-generation detection method was released for Nmap in version 4.50, which is more complete, effective, and accurate than the previous one. This method includes other packets, probes, and tests that make Honeyd's emulation methods significantly less effective. The new file that contains the database is called "nmap-os-db" (Nmap operating system database) and includes several new tests and significant changes to the previous tests and responses.

The new method sends up to 16 TCP, UDP, and ICMP probes to perform 13 different tests, nine derived from the first-generation fingerprinting and four new. These tests are:

- SEQ, TCP sequence generation (similar to the older TSeq)
- OPS, TCP options
- WIN, TCP initial window size
- ECN, TCP explicit congestion notification
- T1-T7 (similar to the older T1-T7)
- U1, UDP packet send to a closed port (similar to the older PU)
- IE, ICMP echo request

Here is the same Windows XP SP1 in the database of the second-generation method used by Nmap.

```
# Windows XP Professional with SP1
Fingerprint Microsoft Windows XP Professional SP1
Class Microsoft | Windows | XP | general purpose
SEQ(SP=82-8C%GCD=1-6%ISR=98-A2%TI=I%II=I%SS=S%TS=0)
OPS(O1=M582NW0NNT00NNS%O2=M582NW0NNT00NNS%O3=M582NW0NNT00%O4=M
582NW0NNT00NNS%O5=M582NW0NNT00NNS%O6=M582NNT00NNS)
WIN(W1=FD5C%W2=FD5C%W3=FD5C%W4=FD5C%W5=FD5C%W6=FD5C)
ECN(R=Y%DF=Y%T=7B-85%TG=80%W=FD5C%O=M582NW0NNS%CC=N%Q=)
T1(R=Y%DF=Y%T=7B-85%TG=80%S=O%A=S+%F=AS%RD=0%Q=)
T2(R=Y%DF=N%T=7B-85%TG=80%W=0%S=Z%A=S%F=AR%O=%RD=0%Q=)
T3(R=Y%DF=Y%T=7B-
85%TG=80%W=FD5C%S=O%A=S+%F=AS%O=M582NW0NNT00NNS%RD=0%Q=)
T4(R=Y%DF=N%T=7B-85%TG=80%W=0%S=A%A=O%F=R%O=%RD=0%Q=)
T5(R=Y%DF=N%T=7B-85%TG=80%W=0%S=Z%A=S+%F=AR%O=%RD=0%Q=)
T6(R=Y%DF=N%T=7B-85%TG=80%W=0%S=A%A=O%F=R%O=%RD=0%Q=)
T7(R=Y%DF=N%T=7B-85%TG=80%W=0%S=Z%A=S+%F=AR%O=%RD=0%Q=)
U1(DF=N%T=7B-
85%TG=80%IPL=38%UN=0%RIPL=G%RID=G%RIPCK=G%RUCK=G%RUD=G)
IE(DFI=S%T=7B-85%TG=80%CD=Z)
```

Consequently, using Nmap 4.5 to detect the operating system for a host emulated by Honeyd, results in either failure or multiple operating-system matches. For instance, if we emulate Microsoft XP SP1 with Honeyd, Nmap 5 will display the message "No exact OS matches for the host."

Is it possible to make the necessary changes to "nmap.prints" file to work well with the second generation Nmap? To answer this we need to analyze the meaning of the results and then try to adapt "nmap.prints" to what new versions of Nmap expect to receive.

TSeq test, the TCP sequence test (now called SEQ), needs the modifications shown in Table 14. We also must change the format for the timestamp from an integer 7, to its equivalent in first-generation, 100HZ. Moreover, we need to respond to the following new tests: TCP ISN sequence predictability index (SP), TCP ISN counter rate (ISR), and shared IP ID sequence Boolean (SS).

First generation	Second generation	Description
Class	-	OS classification or device type
gcd	GCD	TCP ISN greatest common divisor
SI	-	
IPID	TI, CI, II	IP ID sequence generation algorithm
TS	TS	TCP timestamp option algorithm

Table 14.   Modifications in Tseq test

The seven tests from T1 to T7 have the modifications shown in Table 15. The following subtests are new in second generation and do not appear in the first-generation: IP initial time-to-live (T), IP initial time-to-live guess (TG), TCP sequence number (S), TCP reset data checksum (RD), and TCP miscellaneous quirks (Q).

First generation	Second generation	Description
Resp	R	Responsiveness
DF	DF	Do not fragment bit
W	W	Windows size.
ACK	A	TCP acknowledged number
Flags	F	TCP flags
Ops	O	TCP options

Table 15.   Modifications in tests T1–T7

The PU test, a probe sent to a closed UDP port now called U1 needs the modifications shown in Table 16. We need to change the G (good) in second generation to an E (expected) in the results for RID, RIPCK, UCK and DAT tests. Also, the following subtests are new in the second generation and do not appear

in the first generation: IP initial time-to-live (T), IP initial time-to-live guess (TG) and unused port unreachable field non-zero (UN). Three complete tests are entirely new in the second-generation: TCP options (OPS), TCP initial window size (WIN) and ICMP echo (IE).

First generation	Second generation	Description
Resp	R	Responsiveness
DF	DF	Do not fragment bit
TOS	-	Type of Service, removed in new version
IPLEN	IPL	IP total length
RIPTL	RIPL	Returned probe IP total length value
RID	RID	Returned probe IP ID value
RIPCK	RICPK	Integrity of returned probe IP checksum value
UCK	RUCK	Integrity of returned probe UDP checksum
ULEN	-	UDP length
DAT	RUD	Returned data

Table 16.   Modifications in PU test

Considering the equivalent values in both methods, we could make some changes to try to adapt a first generation fingerprint to a new version of Nmap. Unfortunately, experiments with operating-system detection were disappointing, and Nmap's guesses for the operating system were diverse.

To get more success in the operating-system detection, we can change the "default TCP action" to "open," instead of the commonly used "reset" or "block." With this modification the emulated operating system is detected by Nmap with confidence values between 80% and 90%, but the high number of open ports in the host is not a credible configuration and could be suspicious if an attacker inspected it. With this modification, we could also emulate new operating systems not available in the original "nmap.prints" file, like Windows XP

SP3, Windows Server 2008, or Windows 7—all of which could be detected by new versions of Nmap with similar confidence values.

The following are the fingerprints created for these operating systems:

Fingerprint Windows 7
TSeq(Class=TR%gcd=<6%IPID=I%TS=100HZ)
T1(Resp=Y%DF=Y%W=2000%ACK=S++%Flags=AS%Ops=MNWNNT)
T2(Resp=Y%DF=Y%W=0%ACK=S%Flags=AR%Ops=)
T3(Resp=Y%DF=Y%W=0%ACK=O%Flags=AR%Ops=)
T4(Resp=Y%DF=Y%W=0%ACK=O%Flags=R%Ops=)
T5(Resp=Y%DF=Y%W=0%ACK=S++%Flags=AR%Ops=)
T6(Resp=Y%DF=Y%W=0%ACK=O%Flags=R%Ops=)
T7(Resp=Y%DF=Y%W=0%ACK=S++%Flags=AR%Ops=)
PU(DF=N%TOS=0%IPLEN=164%RIPTL=148%RID=E%RIPCK=E%UCK=E%ULEN=134%
DAT=E)

Fingerprint Microsoft Windows Server 2008
Class Microsoft | Windows | 2008 | general purpose
TSeq(Class=TR%gcd=1-6)
T1(Resp=Y%DF=Y%ACK=S++%Flags=AS)
T2(Resp=Y%DF=Y%W=0%ACK=S%Flags=AR%Ops=)
T3(Resp=Y%DF=Y%W=0%ACK=O%Flags=AR%Ops=)
T4(Resp=Y%DF=Y%W=0%ACK=O%Flags=R%Ops=)
T5(Resp=Y%DF=Y%W=0%ACK=S++%Flags=AR%Ops=)
T6(Resp=Y%DF=Y%W=0%ACK=O%Flags=R%Ops=)
T7(Resp=Y%DF=Y%W=0%ACK=S++%Flags=AR%Ops=)
PU(DF=N%IPLEN=164%RIPTL=E%RID=E%RIPCK=E%UCK=E)

Fingerprint Windows XP SP3
TSeq(Class=TR%IPID=I%TS=0)
T1(DF=Y%W=FAF0%ACK=S++%Flags=AS%Ops=MNWNNT)
T2(Resp=Y%DF=N%W=0%ACK=S%Flags=AR%Ops=)
T3(Resp=Y%DF=Y%W=FAF0%ACK=S++%Flags=AS%Ops=MNWNNT)
T4(DF=N%W=0%ACK=O%Flags=R%Ops=)
T5(DF=N%W=0%ACK=S++%Flags=AR%Ops=)
T6(DF=N%W=0%ACK=O%Flags=R%Ops=)

66

T7(DF=N%W=0%ACK=S++%Flags=AR%Ops=)

PU(DF=N%TOS=0%IPLEN=B0%RIPTL=148%RID=E%RIPCK=E%UCK=E%ULEN=134%

DAT=E)

Until there is a patch or a new Honeyd version that works better with second-generation Nmap, in order to simulate different operating systems we have to rely on a credible port configuration, opening common, or well-known ports for the operating system we want to emulate.

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF REFERENCES

[1]   N. Provos, "Developments of the virtual honeyd honeypot." Honeyd.org. Available: http://www.honeyd.org/.

[2]   The Honeynet Project, *Know Your Enemy: Learning about Security Threats* (2nd ed.). Boston, MA: Addison-Wesley, 2004.

[3]   N. Provos and T. Holz, *Virtual Honeypots: From Botnet Tracking to Intrusion Detection.* Boston, MA: Addison-Wesley Professional, 2008.

[4]   N. Krawetz, "Anti-honeypot technology," *IEEE Security & Privacy*, pp. 76–79, January/February 2004.

[5]   G. Giuseppini, *Microsoft Log Parser Toolkit,* Waltham, MA: Syngress Publishing, 2005.

[6]   B. McCarty, "The honeynet arms race," *IEEE Security & Privacy,* pp. 79–82, November/December 2003.

[7]   R. Grimes, *Honeypots for Windows.* Berkeley, CA: A-Press, 2005.

[8]   N. Rowe, "Measuring the effectiveness of honeypot counter-counterdeception," *Proc. 39th Hawaii International Conference on Systems Sciences*, Poipu, HI, January 2006.

[9]   B. Duong, "Comparisons of attacks on honeypots with those on real networks," M.S. thesis, Naval Postgraduate School, 2006.

[10]   S. Lim, "Assessing the effect of honeypots on cyber-attackers," M.S. thesis, Naval Postgraduate School, 2006.

[11]   L. Spitzner, *Honeypots: Tracking Hackers.* Boston, MA: Addison-Wesley, 2003.

[12]   Nmap.org, "TCP/IP Fingerprinting methods supported by Nmap," Chapter 8. Remote OS Detection. Available: http://nmap.org/book/osdetect-methods.html.

THIS PAGE INTENTIONALLY LEFT BLANK

INITIAL DISTRIBUTION LIST

1.   Defense Technical Information Center
        Ft. Belvoir, Virginia

2.   Dudley Knox Library
        Naval Postgraduate School
        Monterey, California

3.   Dr. Neil C. Rowe
        Naval Postgraduate School
        Monterey, California

4.   Mr. Daniel F. Warren
        Naval Postgraduate School
        Monterey, California

5.   LCDR Erwin E. Frederick
        Chilean Navy
        Valparaiso, Chile